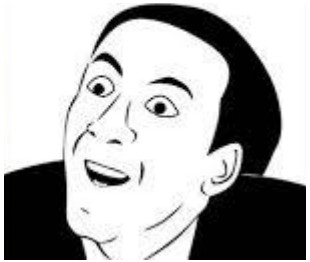
CNN models

ILSVRC : ImageNet

- ImageNet Large Scale Visual Recognition Challenge
- 1000 classes 1.2M images
- 李飛飛教授
- Official: 2010~~2017
- ImageNet 1000 classes

恩特布山犬

阿彭策爾山犬



History

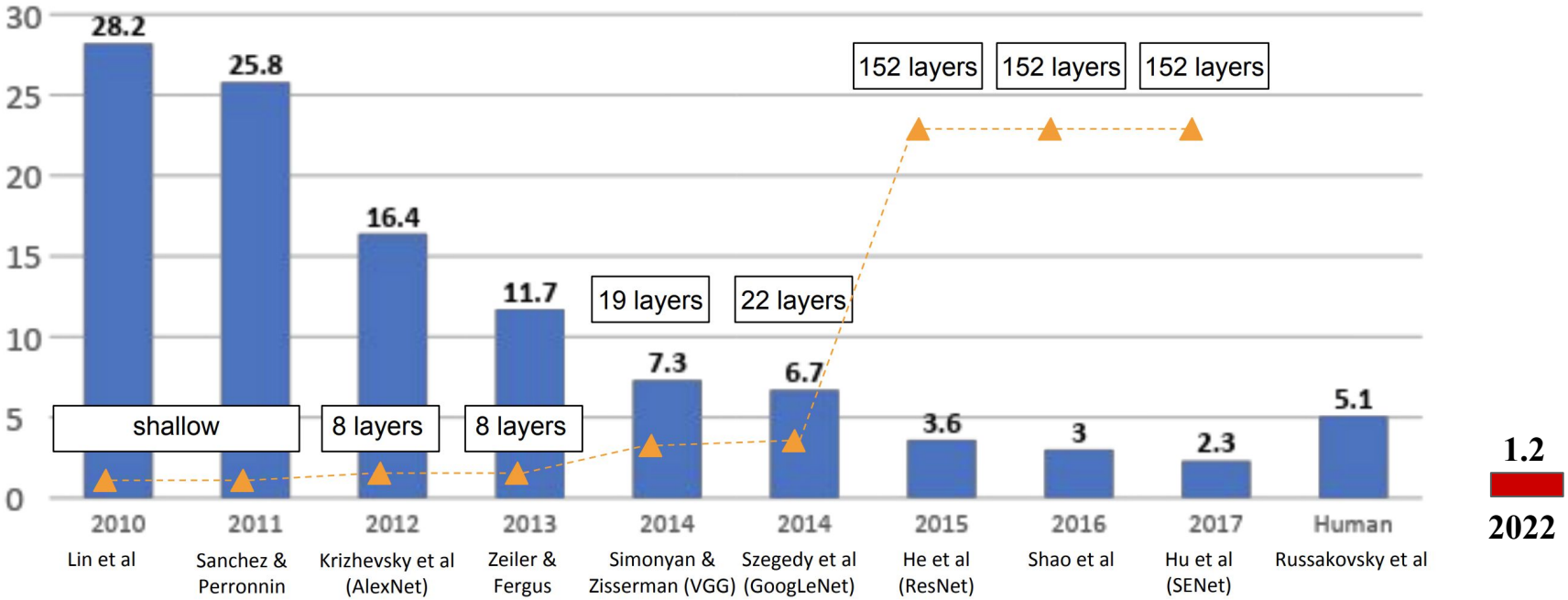
- 1989: **Yann LeCun** BP for CNN
- 1993: LeNet
- 2012: GPU for CNN - AlexNet
- 2014: GoogleNet, VGG
- 2015: ResNet
- 2017: SENet

FUSION

ImageNet

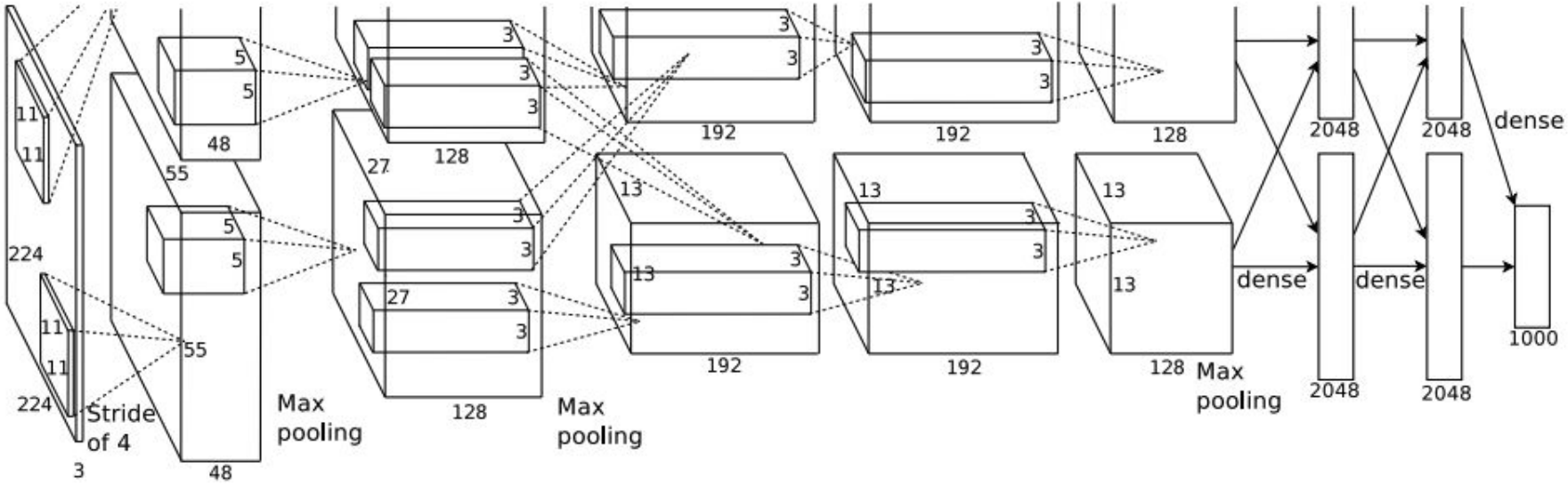
- ~ 2022 9/5: [leader board]

Top 5 Error Rate (%)↓



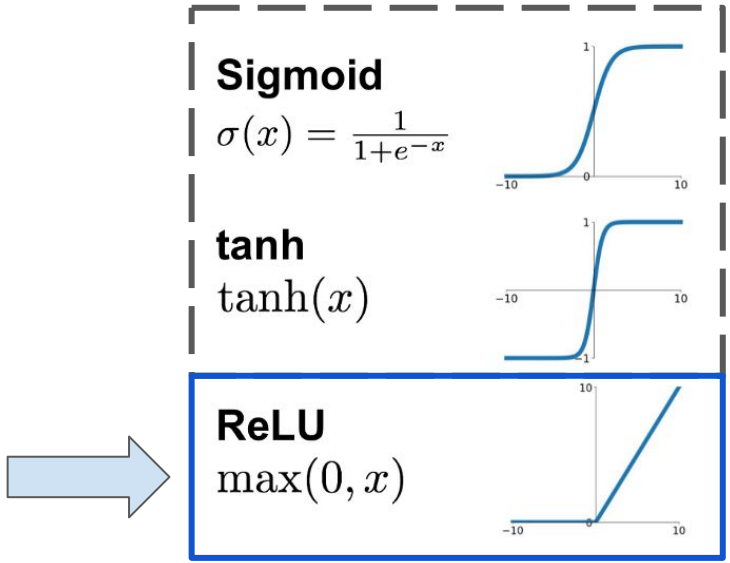
AlexNet (2012)

ImageNet Classification with Deep Convolutional Neural Networks



AlexNet (2012)

- Multi-GPUs
- ReLU
  - faster convergence
  - lower gradient vanishing
- Reduce overfitting
  - Max pooling
  - Data augmentation
  - Dropout



AlexNet (2012)

PyTorch

```
class AlexNet(nn.Module):
    def __init__(self, num_classes=1000):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
```

Encoder

```
self.flatten = nn.Flatten()
self.classifier = nn.Sequential(
    nn.Dropout(),
    nn.Linear(256 * 6 * 6, 4096),
    nn.ReLU(),
    nn.Dropout(),
    nn.Linear(4096, 4096),
    nn.ReLU(),
    nn.Linear(4096, num_classes),
)
```

FUSION

AlexNet (2012)

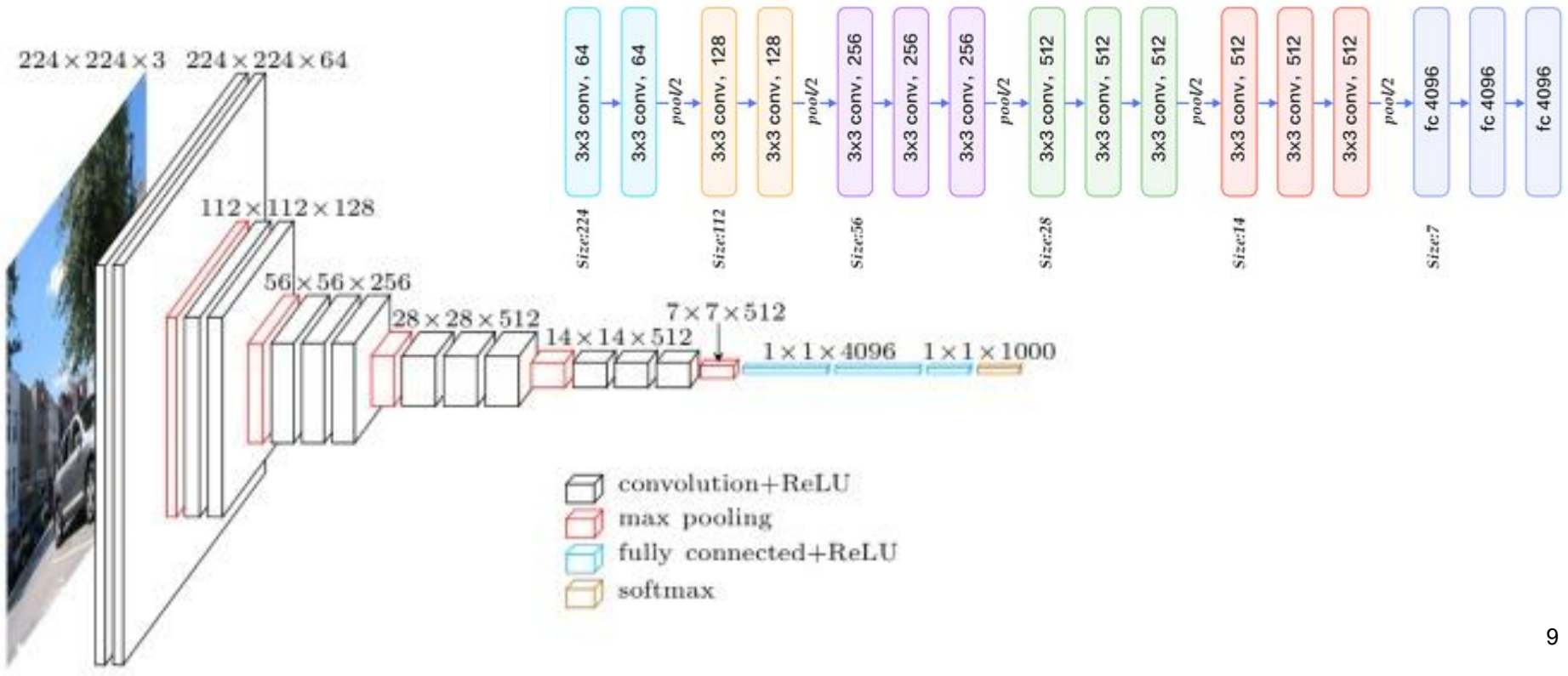
TensorFlow

```
model = Sequential([
    tf.keras.Input((224, 224, 3)),
    Conv2D(96, (11,11), strides=(4,4), activation='relu'),
    MaxPooling2D(pool_size=(3,3), strides=(2,2)),
    Conv2D(256, (5,5), strides=(1,1), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(3,3), strides=(2,2)),
    Conv2D(384, (3,3), strides=(1,1), padding='same',activation='relu'),
    Conv2D(384, (3,3), strides=(1,1), padding='same',activation='relu'),
    Conv2D(256, (3,3), strides=(1,1), padding='same',activation='relu'),
    MaxPooling2D(pool_size=(3,3), strides=(2,2)),
    Flatten(),
    Dense(4096, activation='relu'),
    Dropout(0.5),
    Dense(4096, activation='relu'),
    Dropout(0.5),
    Dense(1000, activation='softmax'),
])
```

FUSION

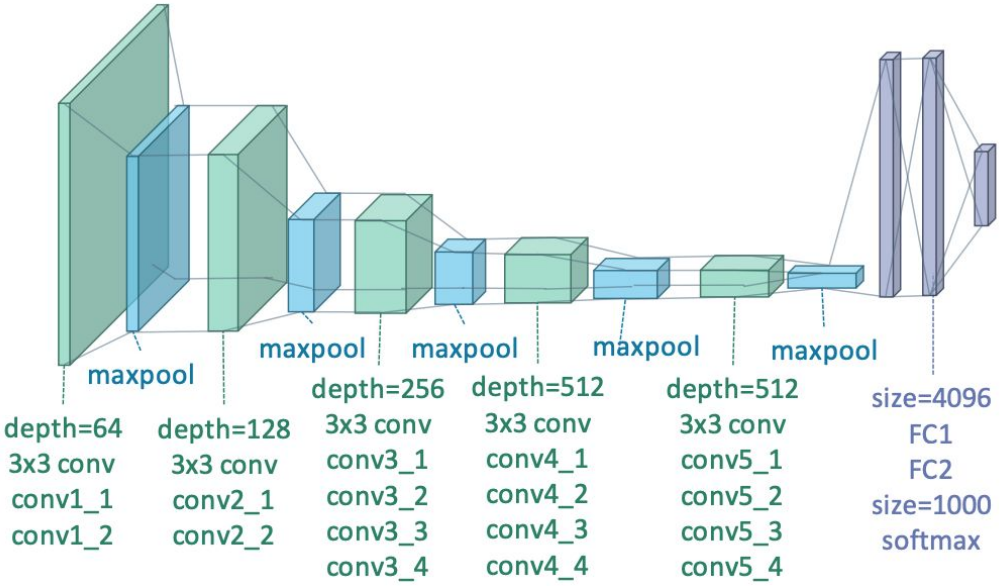
VGG (2014)

Very Deep Convolutional Networks for Large-Scale Image Recognition



VGG

VGG16, VGG19



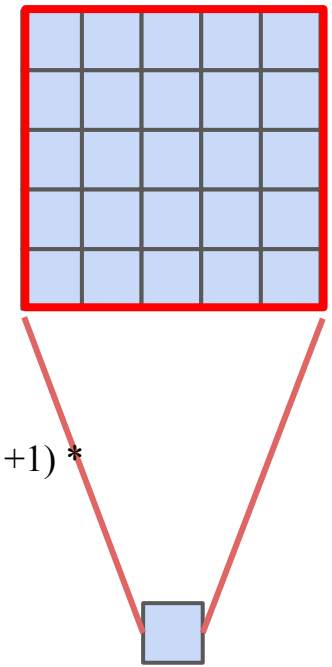
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

source: https://www.csie.ntu.edu.tw/~yvchen/f105-adl/doc/161103_ConvolutionalNN.pdf

Receptive Fields (視野, 感知野)

- 3x3 kernels
  - Same **receptive fields** with less parameters
  - More non-linearity (activation function)

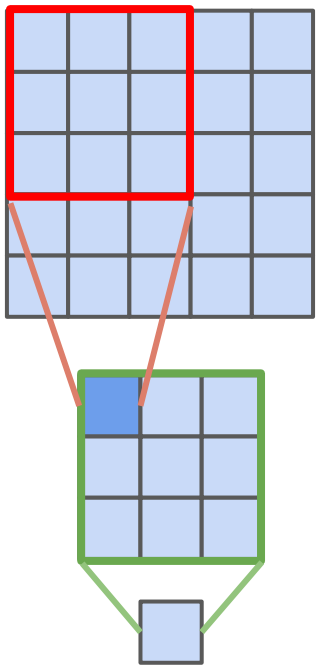
If C = 1



(C, 5, 5)

(C', 3, 3)

(C", 1, 1)



5x5 filter * 1

- parameters: (5*5*1+1) * 1 = **26**
- activation: **1**

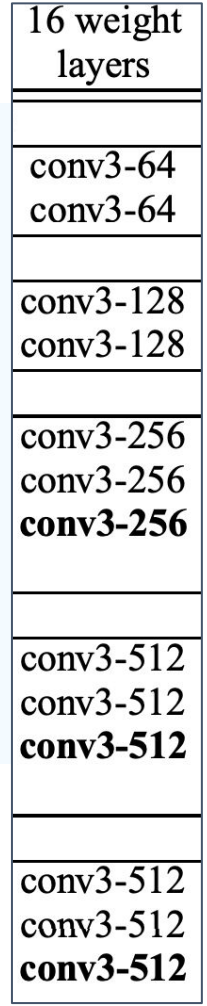
3x3 filter * 2

- parameters: (3*3*1+1) * 1 = 10
  10 * 2 = **20**
- activation: **2**

VGG16: PyTorch Code

```
# Block 1
nn.Conv2d(3, 64, kernel_size=3, padding=1),
nn.ReLU(),
nn.Conv2d(64, 64, kernel_size=3, padding=1),
nn.ReLU(),
nn.MaxPool2d(kernel_size=2, stride=2),
# Block 2
nn.Conv2d(64, 128, kernel_size=3, padding=1),
nn.ReLU(),
nn.Conv2d(128, 128, kernel_size=3, padding=1),
nn.ReLU(),
nn.MaxPool2d(kernel_size=2, stride=2),
# Block 3
nn.Conv2d(128, 256, kernel_size=3, padding=1),
nn.ReLU(),
nn.Conv2d(256, 256, kernel_size=3, padding=1),
nn.ReLU(),
nn.Conv2d(256, 256, kernel_size=3, padding=1),
nn.ReLU(),
nn.MaxPool2d(kernel_size=2, stride=2),
```

```
# Block 4
nn.Conv2d(256, 512, kernel_size=3, padding=1),
nn.ReLU(),
nn.Conv2d(512, 512, kernel_size=3, padding=1),
nn.ReLU(),
nn.Conv2d(512, 512, kernel_size=3, padding=1),
nn.ReLU(),
nn.MaxPool2d(kernel_size=2, stride=2),
# Block 5
nn.Conv2d(512, 512, kernel_size=3, padding=1),
nn.ReLU(),
nn.Conv2d(512, 512, kernel_size=3, padding=1),
nn.ReLU(),
nn.Conv2d(512, 512, kernel_size=3, padding=1),
nn.ReLU(),
nn.MaxPool2d(kernel_size=2, stride=2),
```



FUSION

VGG16

TensorFlow

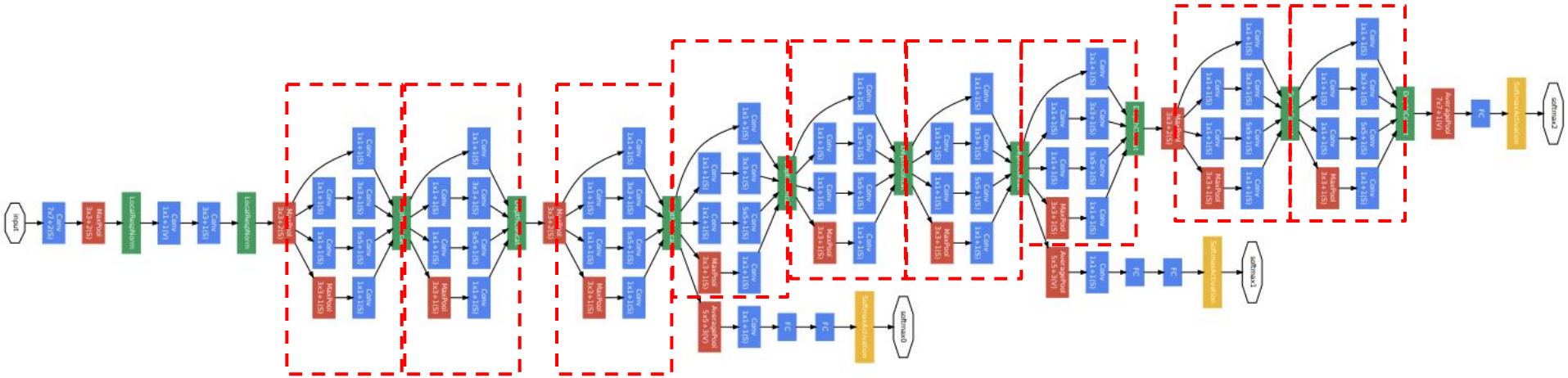
```
tf.keras.Input((224, 224, 3)),
Conv2D(64, (3,3), padding='same', activation='relu'),
Conv2D(64, (3,3), padding='same', activation='relu'),
MaxPooling2D(pool_size=(2,2)),
Conv2D(128, (3,2), padding='same', activation='relu'),
Conv2D(128, (3,3), padding='same', activation='relu'),
MaxPooling2D(pool_size=(2,2)),
Conv2D(256, (3,3), padding='same', activation='relu'),
Conv2D(256, (3,3), padding='same', activation='relu'),
Conv2D(256, (3,3), padding='same', activation='relu'),
MaxPooling2D(pool_size=(2,2)),
Conv2D(512, (3,3), padding='same', activation='relu'),
Conv2D(512, (3,3), padding='same', activation='relu'),
Conv2D(512, (3,3), padding='same', activation='relu'),
MaxPooling2D(pool_size=(2,2)),
Conv2D(512, (3,3), padding='same', activation='relu'),
Conv2D(512, (3,3), padding='same', activation='relu'),
Conv2D(512, (3,3), padding='same', activation='relu'),
MaxPooling2D(pool_size=(2,2)),
Flatten(),
Dense(4096,activation='relu'),
Dropout(0.5),
Dense(4096,activation='relu'),
Dropout(0.5),
Dense(1000,activation='softmax'),
```

16 weight layers
conv3-64 conv3-64
conv3-128 conv3-128
conv3-256 conv3-256 **conv3-256**
conv3-512 conv3-512 **conv3-512**
conv3-512 conv3-512 **conv3-512**

FUSION

GoogLeNet

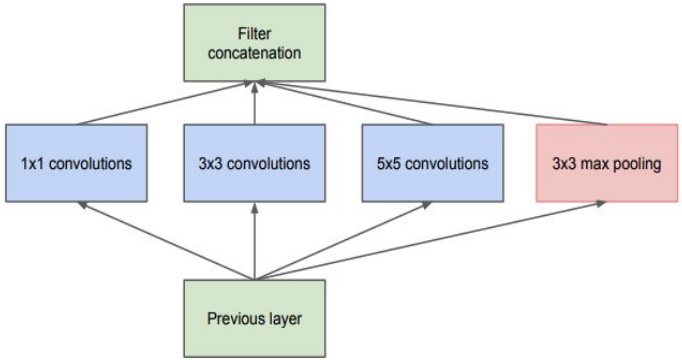
- Going Deeper with Convolutions
- a.k.a InceptionV1

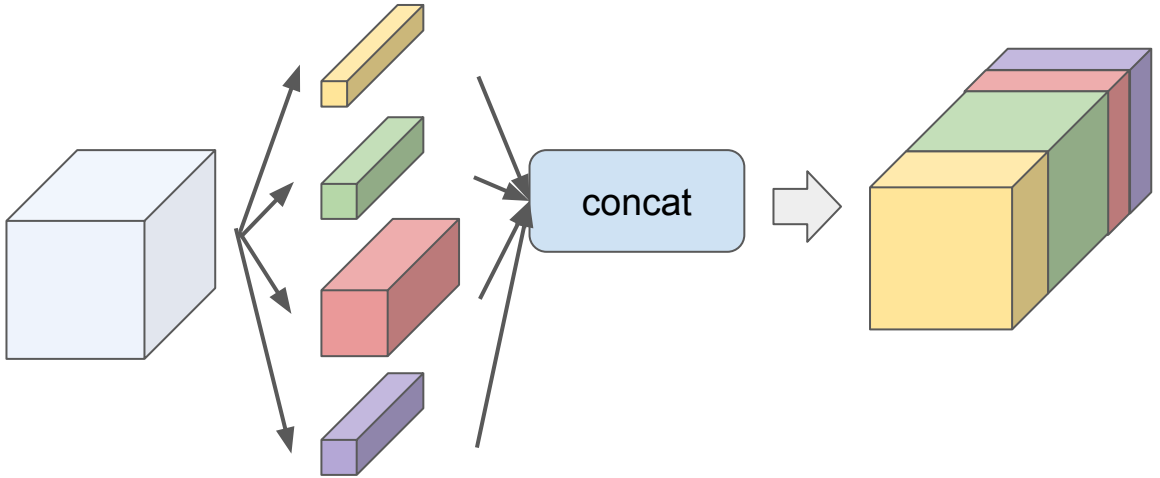


FUSION

GoogLeNet

- Inception module
  - Which **kernel size** is better?
  - Use them all

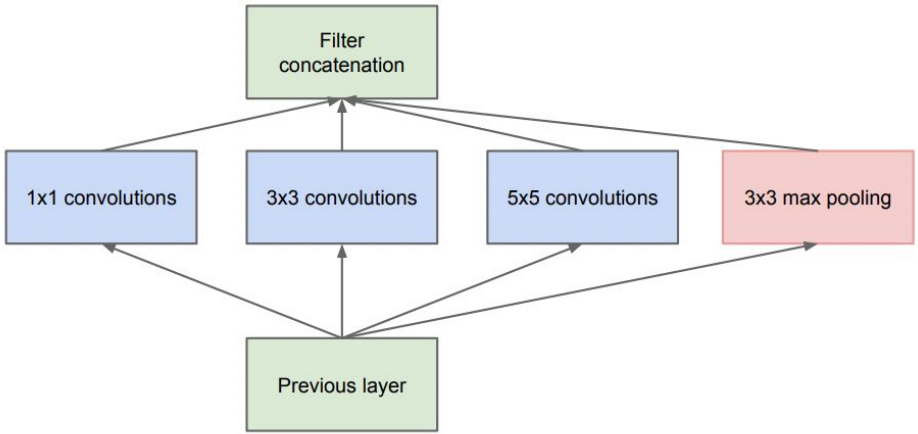




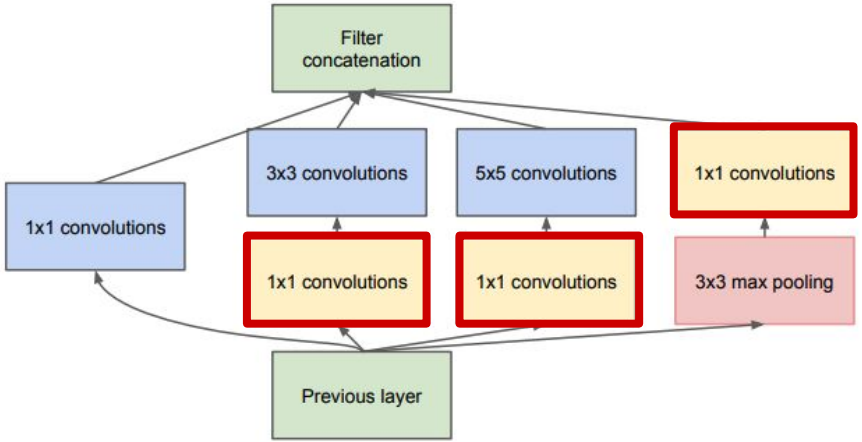
FUSION

GoogLeNet

1x1 convolution: reduce channel dimension (compress)

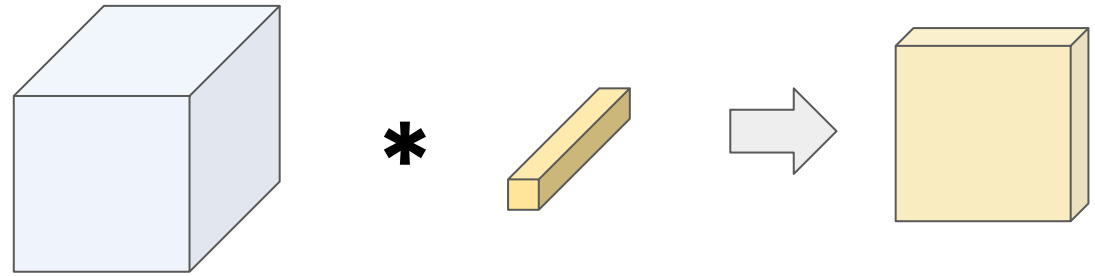


(a) Inception module, naïve version

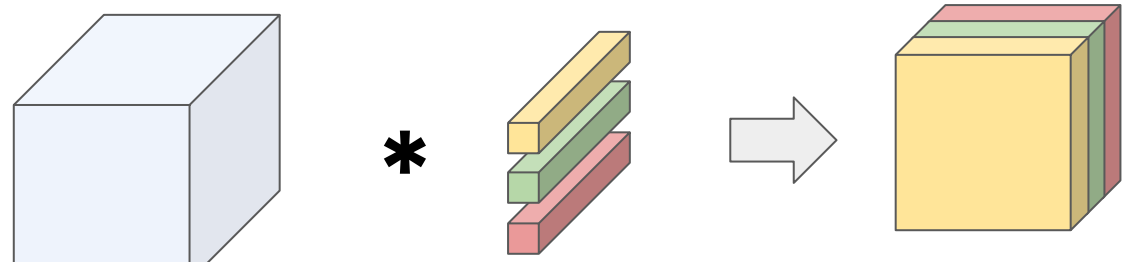


1x1 Convolution (PointwiseConv.)

- Change output channel dimension
  - Reduce ↓
  - Expand ↑
- Keep H, W size (no padding)
- More Non-linearity (activation)

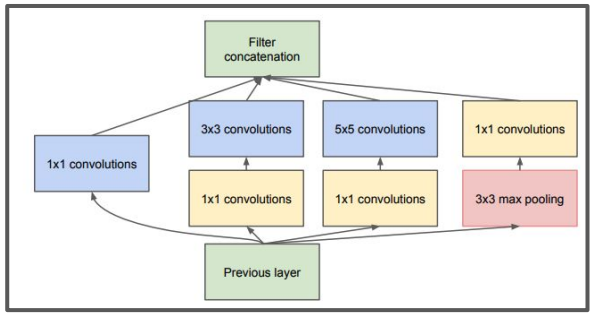


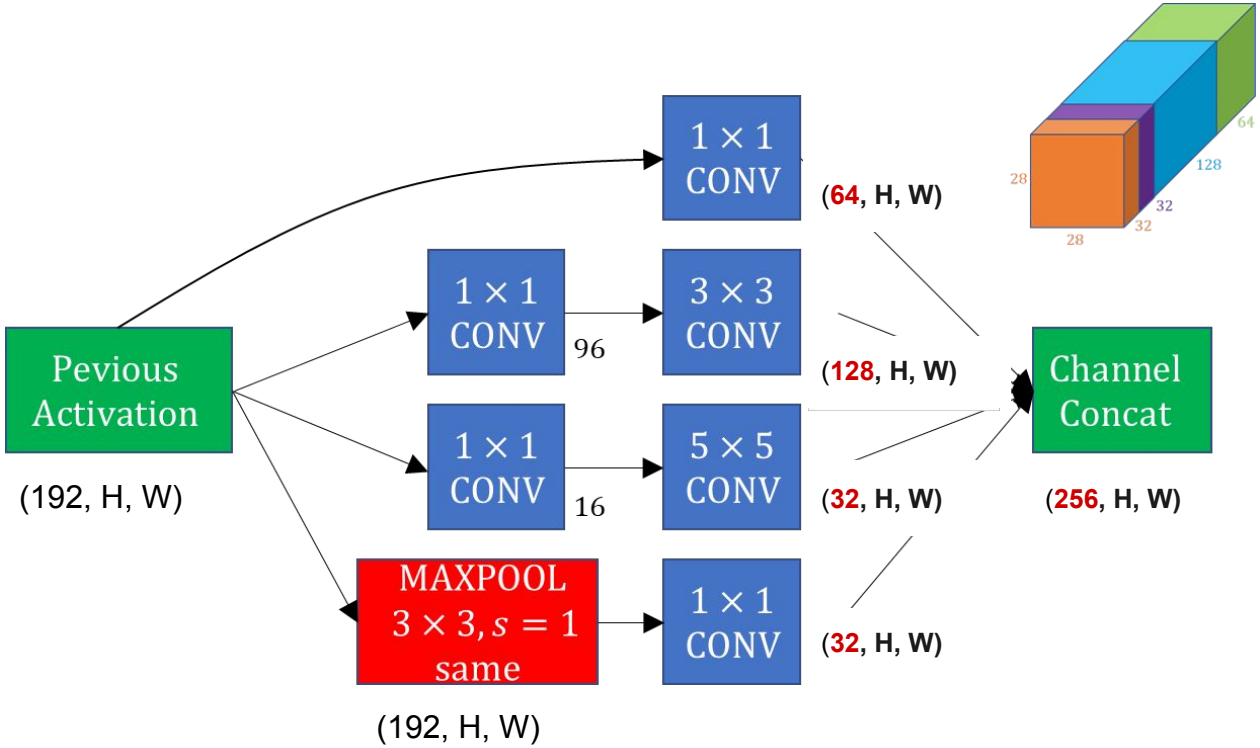
(C_{in}, H, W) (C_{in}, 1, 1) (1, H, W)



(C_{in}, H, W) (C_{in}, 1, 1) x **C** (**C**, H, W)

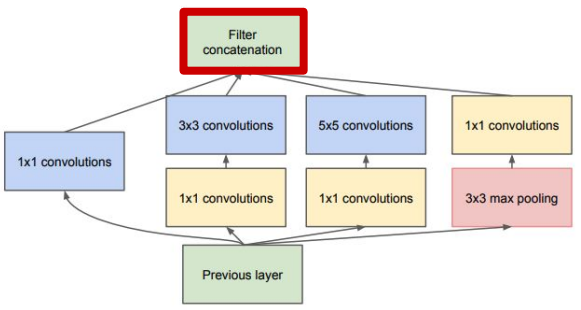
Feature Concatenate





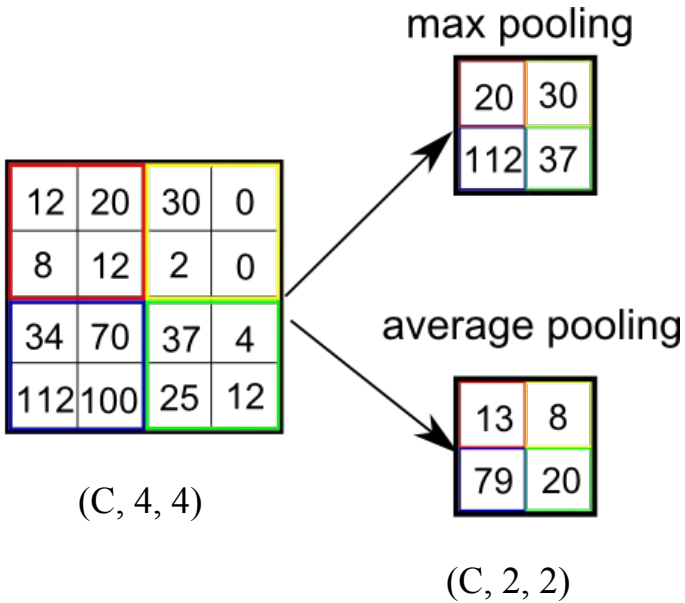
```python
class InceptionModule(nn.Module):
    def __init__(
        self,
        in_channels,
        ch1x1,
        ch3x3red, ch3x3,
        ch5x5red, ch5x5,
        pool_proj):
        super().__init__()
        # branch 1x1
        self.branch1 = BasicConv2d(in_channels, ch1x1, kernel_size=1)
        # branch 3x3
        self.branch2 = nn.Sequential(
            BasicConv2d(in_channels, ch3x3red, kernel_size=1), # reduce chanels by 1x1 conv
            BasicConv2d(ch3x3red, ch3x3, kernel_size=3, padding=1)
        )
        # branch 5x5
        self.branch3 = nn.Sequential(
            BasicConv2d(in_channels, ch5x5red, kernel_size=1), # reduce chanels by 1x1 conv
            BasicConv2d(ch5x5red, ch5x5, kernel_size=5, padding=2)
        )
        # branch Pool
        self.branch4 = nn.Sequential(
            nn.MaxPool2d(kernel_size=3, stride=1, padding=1, ceil_mode=True),
            BasicConv2d(in_channels, pool_proj, kernel_size=1)
        )
```

```python
    def forward(self, x):
        branch1 = self.branch1(x)
        branch2 = self.branch2(x)
        branch3 = self.branch3(x)
        branch4 = self.branch4(x)
        # concatenate feature dims
        outputs = torch.cat([branch1, branch2, branch3, branch4], dim=1)
        return outputs
```



PyTorch FUSION

Average Pooling

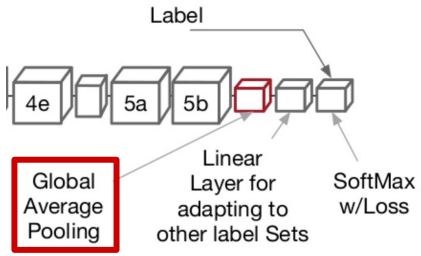


PyTorch

```
torch.nn.MaxPool2d(kernel_size=2)
torch.nn.AvgPool2d(kernel_size=2)
```

FUSION

GAP: Global Average Pooling



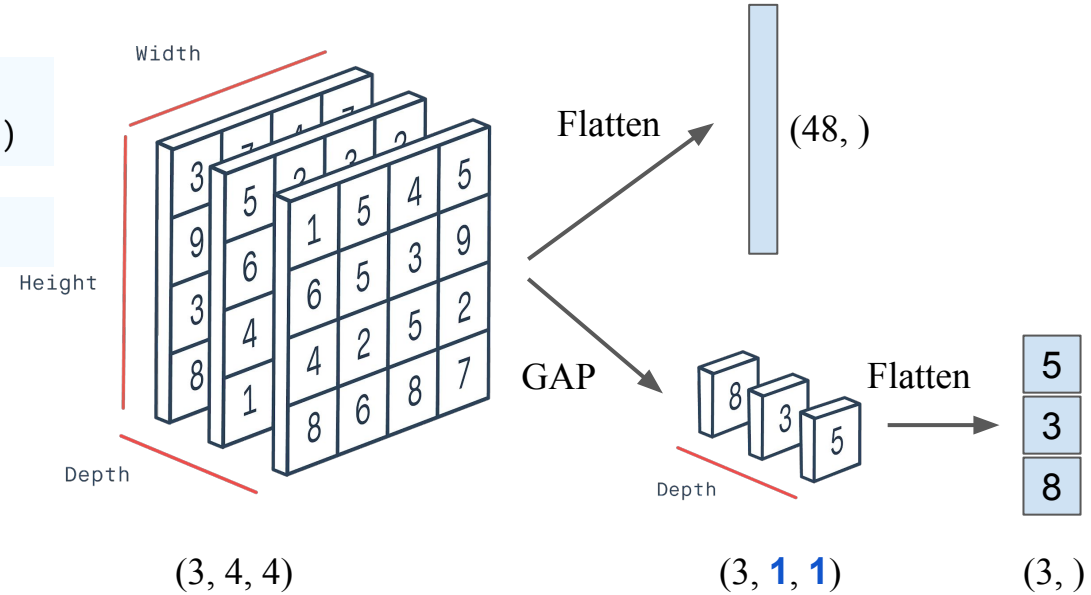
- Replace Flatten layer

PyTorch

```
torch.nn.AdaptiveAvgPool2d(1)
torch.nn.AdaptiveAvgPool2d((1, 1))
```

```
nn.Flatten()
```

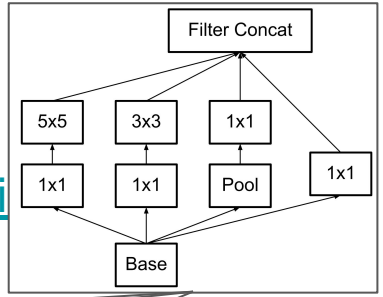
```
nn.Sequential(
    nn.AdaptiveAvgPool2d(1),
    nn.Flatten(),
)
```

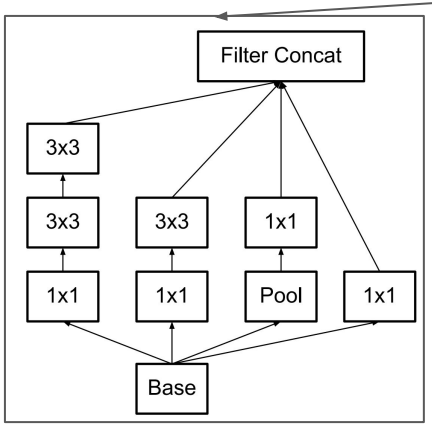


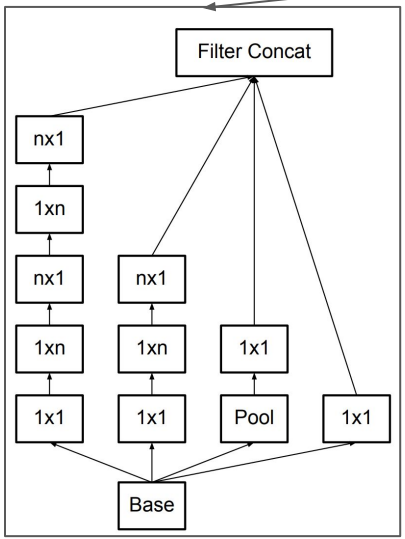
source

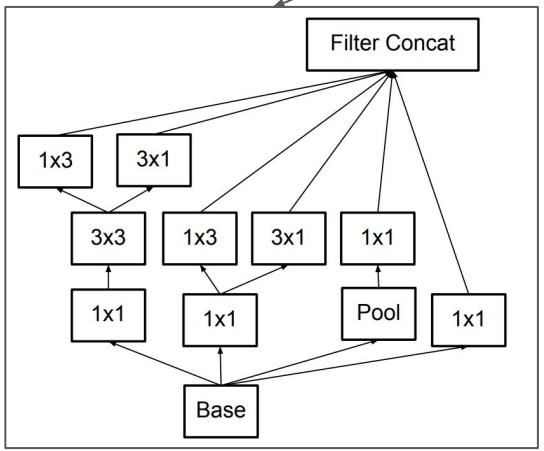
More Inception

- InceptionV2, V3
- [Rethinking the Inception Architecture for Computer Visi]





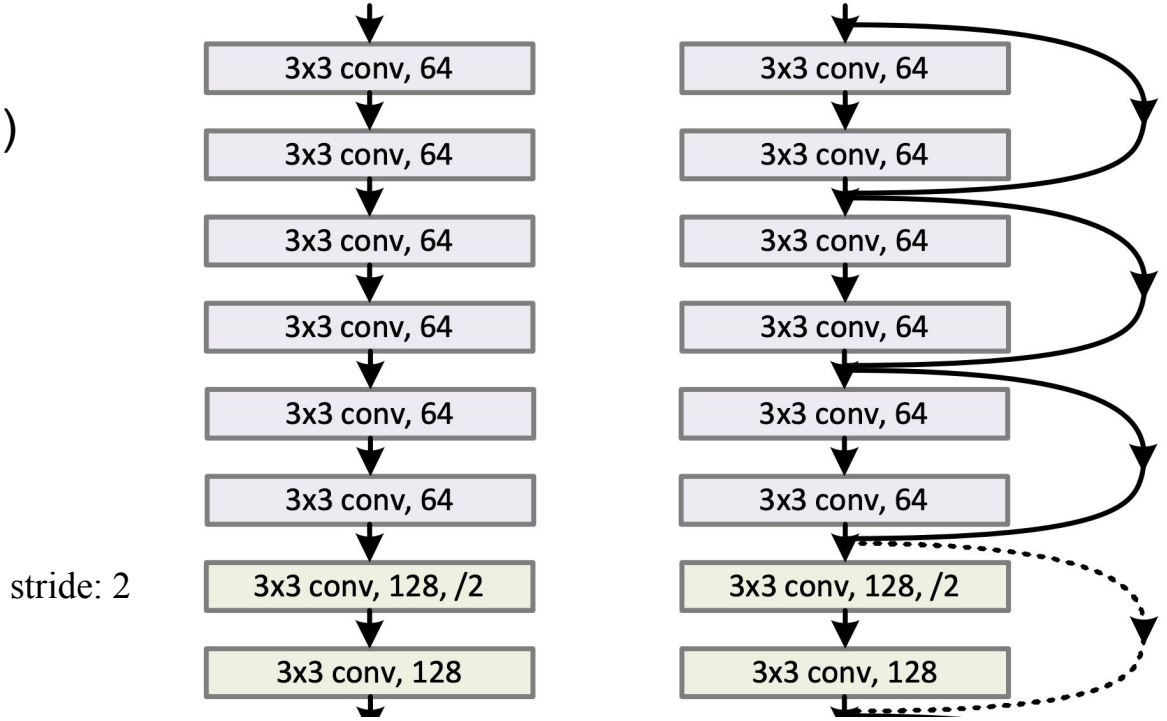




FUSION

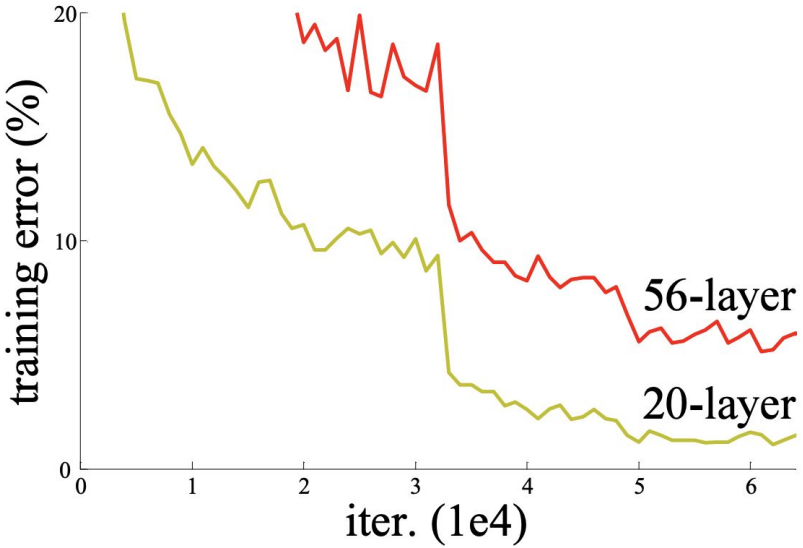
ResNet

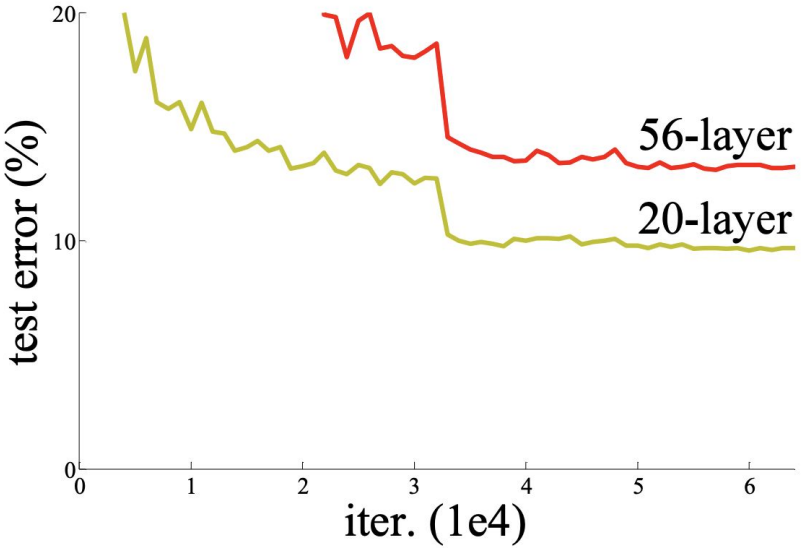
- Deep Residual Learning for Image Recognition (2015)
- 152 layers
- Residual Block (殘差)
- Skip connection



ResNet

- What happened?
- Deeper is Better?
- Not overfitting!

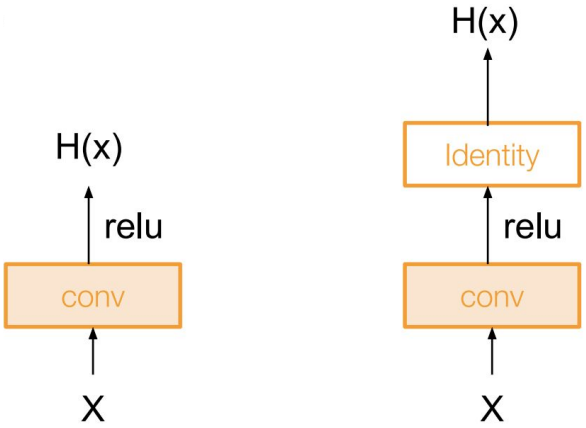




FUSION

ResNet

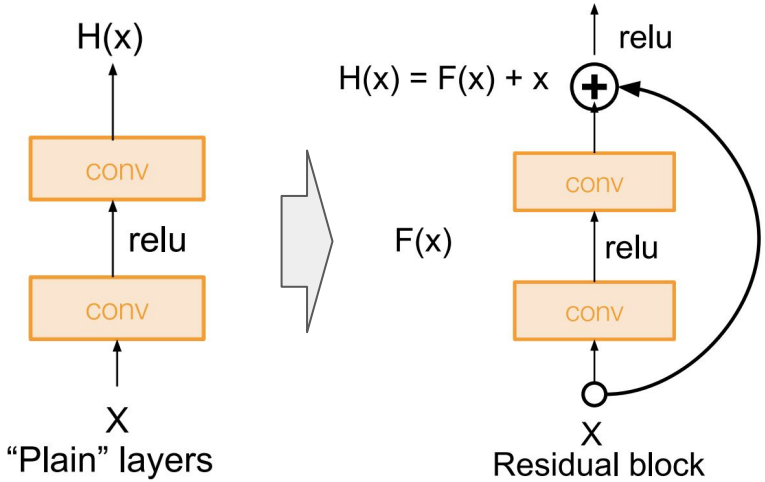
- Deep models have more power than shallower model
- Deep models are **harder** to optimize
- What should the deeper model learn to be at least as good as the shallower model?
  - Identity:
    - output = ***Identity***(input)
    - input = output
    - hard to learn
    - ≈ 装忙



source: http://cs231n.stanford.edu/slides/2020/lecture_9.pdf

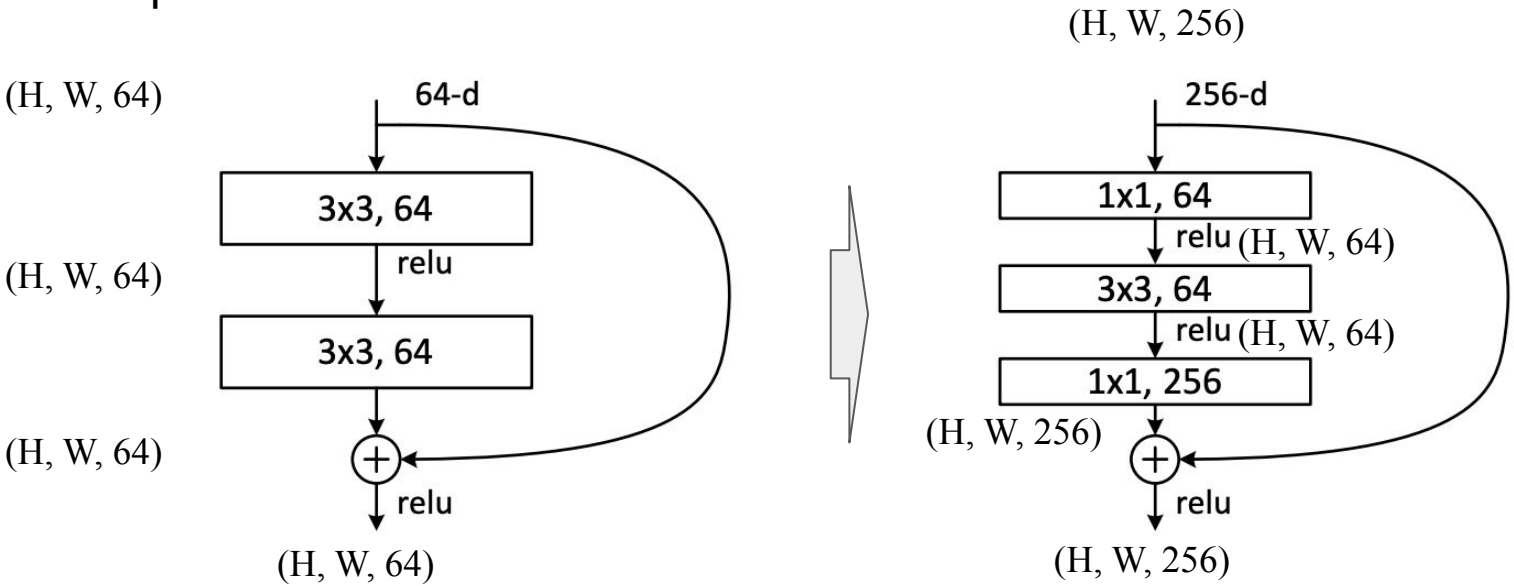
ResNet

- From learning **H(x)**
- To learning **H(x) - x** , (F(x))
- Results:
  - if F(x) is large: these layers are useful
  - if F(x) = 0: these layers are useless



ResNet

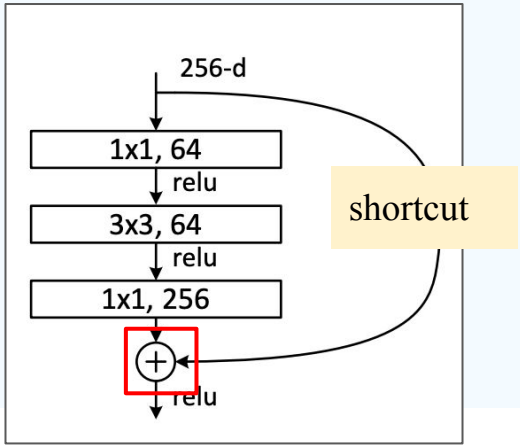
- **Bottleneck** for deeper ResNet (ResNet50, 101, 152)
- Improve computing efficiency
- Less parameters



Residual Block

```
class IdentityBlock(nn.Module):
    def __init__(self, in_dims, out_dims, kernel_size, stride=1):
        super().__init__()
        filters1, filters2, filters3 = out_dims
        self.conv1 = BasicConv2d(in_dims, filters1, kernel_size=1, stride=stride,
                                 bn=True)
        self.conv2 = BasicConv2d(filters1, filters2, kernel_size, padding='same', bn=True)
        self.conv3 = BasicConv2d(filters2, filters3, kernel_size=1, bn=True, activation=False)
        self.relu = nn.ReLU()

    def forward(self, x):
        shortcut = x
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x += shortcut # Add (x, shortcut) tensor
        x = self.relu(x)
        return x
```



FUSION

ResNet

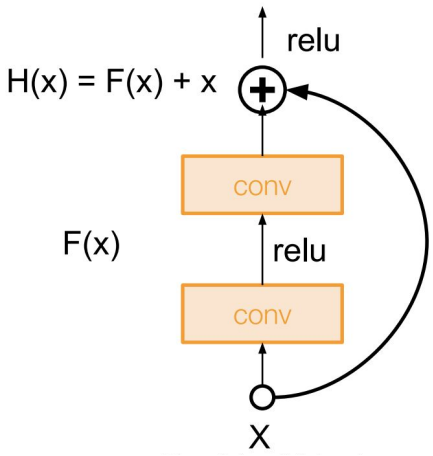
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Basic Residual Block

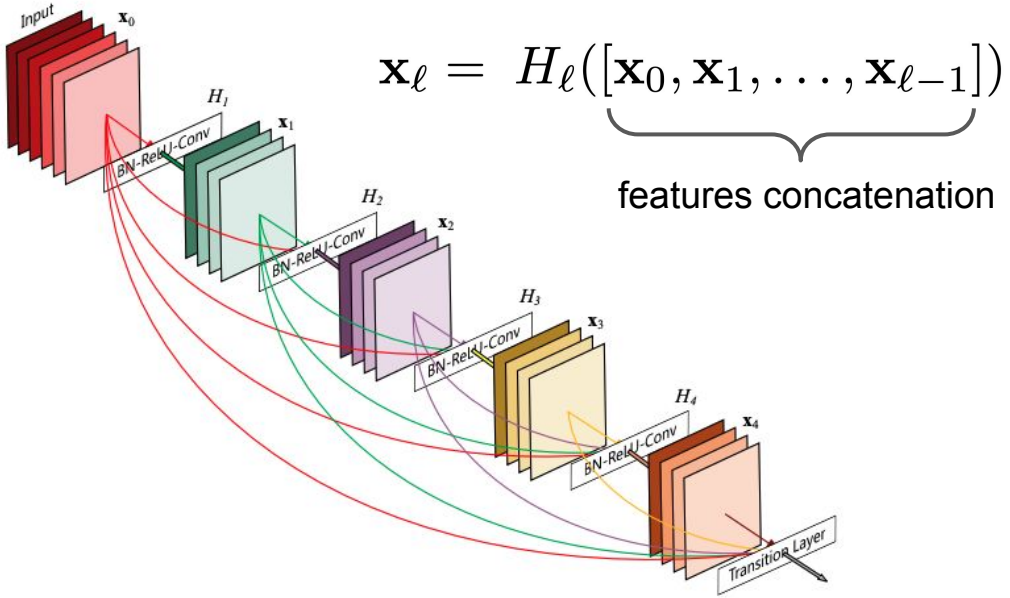
Bottleneck Residual Block

DenseNet

Densely Connected Convolutional Networks (2016)



ResNet



DenseNet

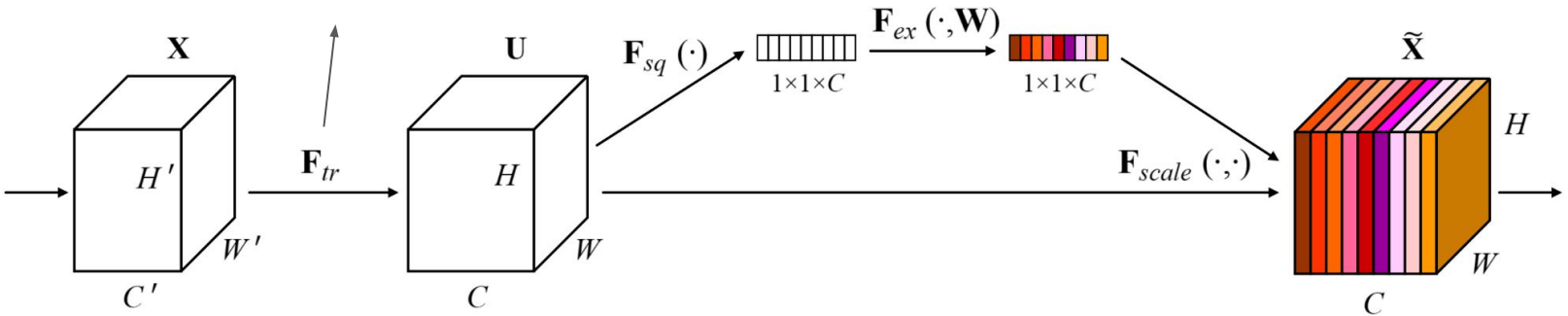
FUSION

SENet

Squeeze-and-Excitation Networks (2017)

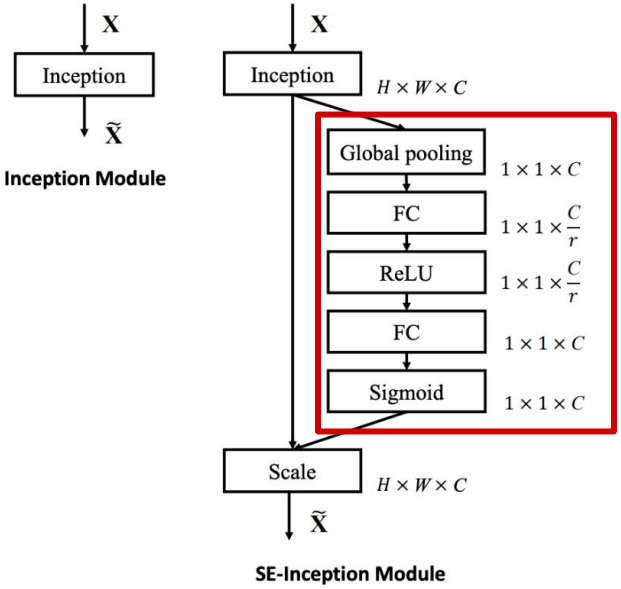
Channel Attention: compute channel importance

Some layers



(C', H', W') (C, H, W) (C, H, W)

Squeeze-and-Excitation Networks (2017)



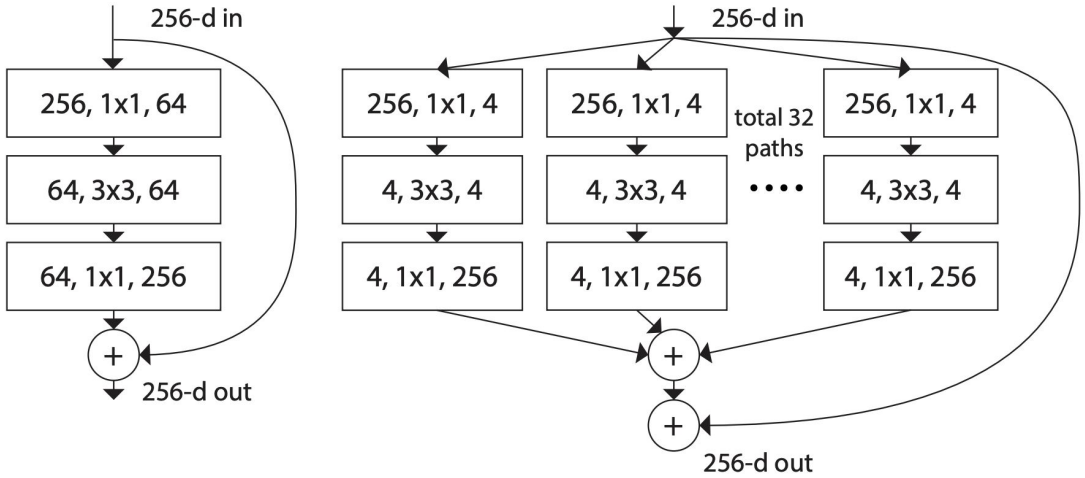

```python
class SEModule(nn.Module):
    def __init__(self, cin, ratio=16):
        super().__init__()
        cout = int(cin / ratio)
        self.gate = nn.Sequential(
            nn.Conv2d(cin, cout, kernel_size=1),
            nn.ReLU(),
            nn.Conv2d(cout, cin, kernel_size=1),
            nn.Sigmoid(),
        )
    def forward(self, inputs):
        x = inputs.mean((2, 3), keepdim=True)
        x = self.gate(x)
        return inputs * x
```

FUSION

ResNeXt

Aggregated Residual Transformations for Deep Neural Networks (2016)

- Improve network ability: increase **depth** and increase **width**
- ResNeXt increasing **cardinality** (# branches in a block)



ResNet ResNeXt

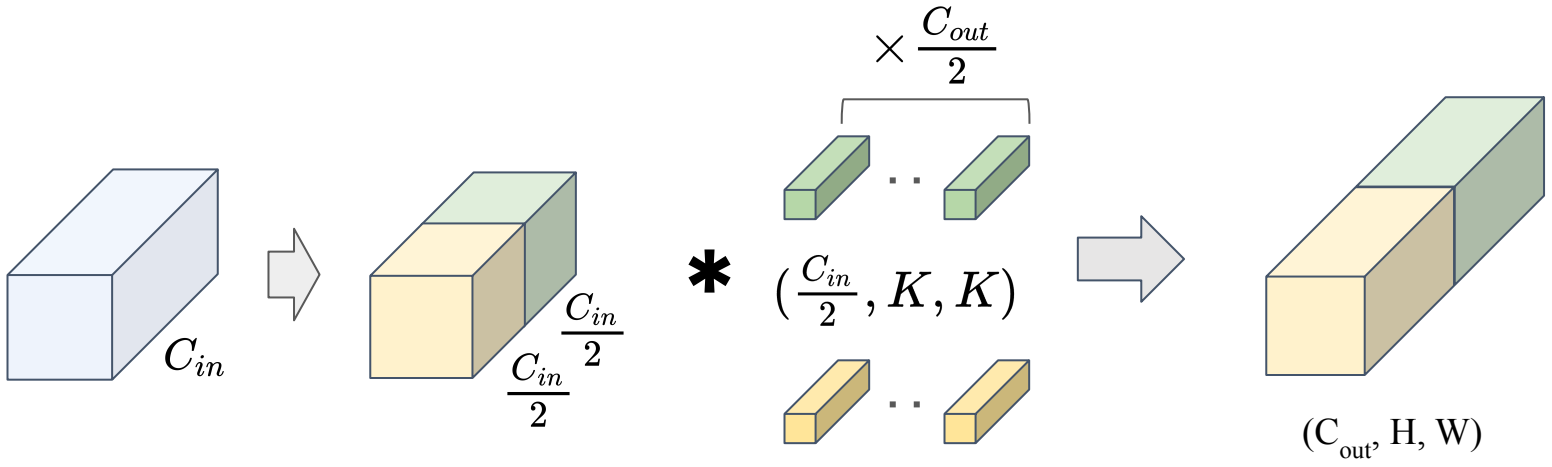
FUSION

Group Convolution

PyTorch

torch.nn.Conv2d(C_{in}, C_{out}, groups=2)

[Youtube]



C_{in} % groups == 0

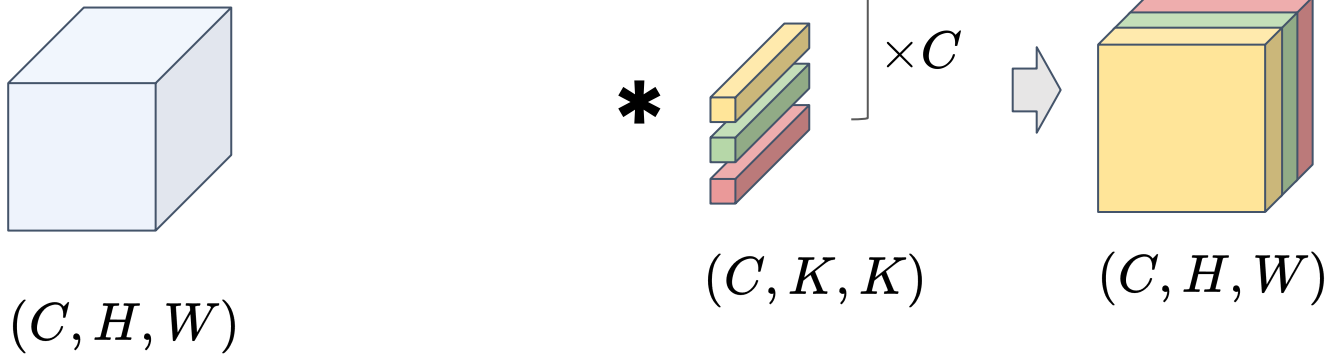
C_{out} % groups == 0

FUSION

Depthwise Conv.

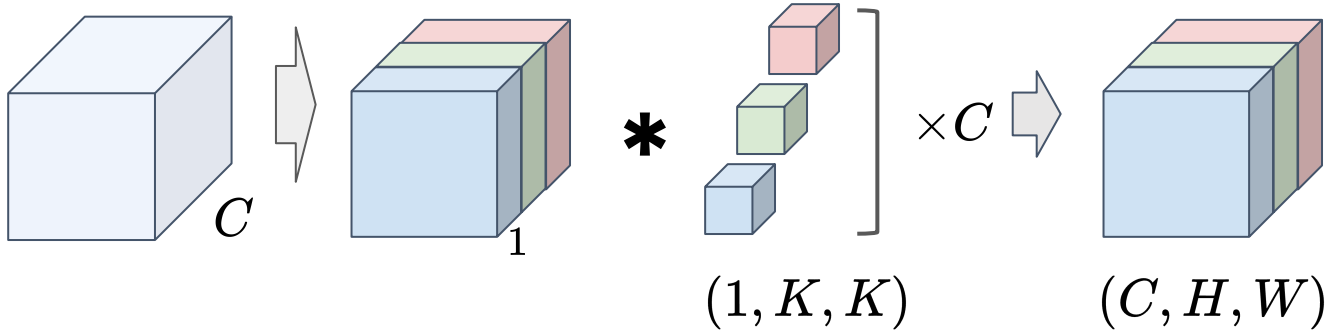
of parameters

Basic Conv.



C x C x K x K

Depthwise Conv.



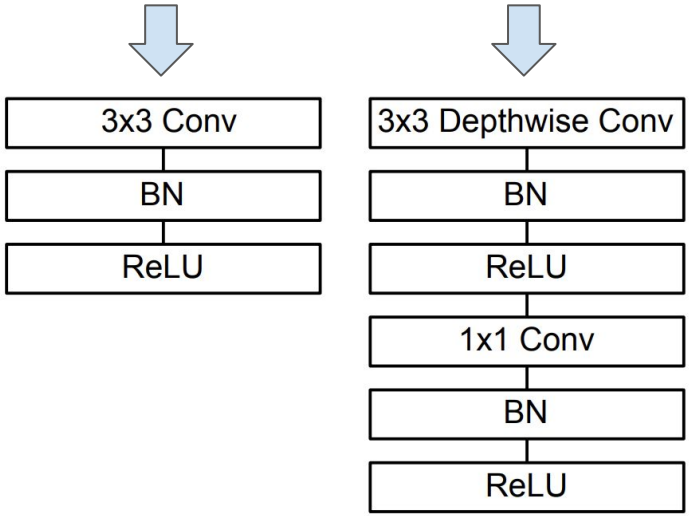
C x 1 x K x K

FUSION

MobileNet

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications (2017)

- Basic conv. → **Depthwise** conv. + **Pointwise** conv.
- Faster andd smaller

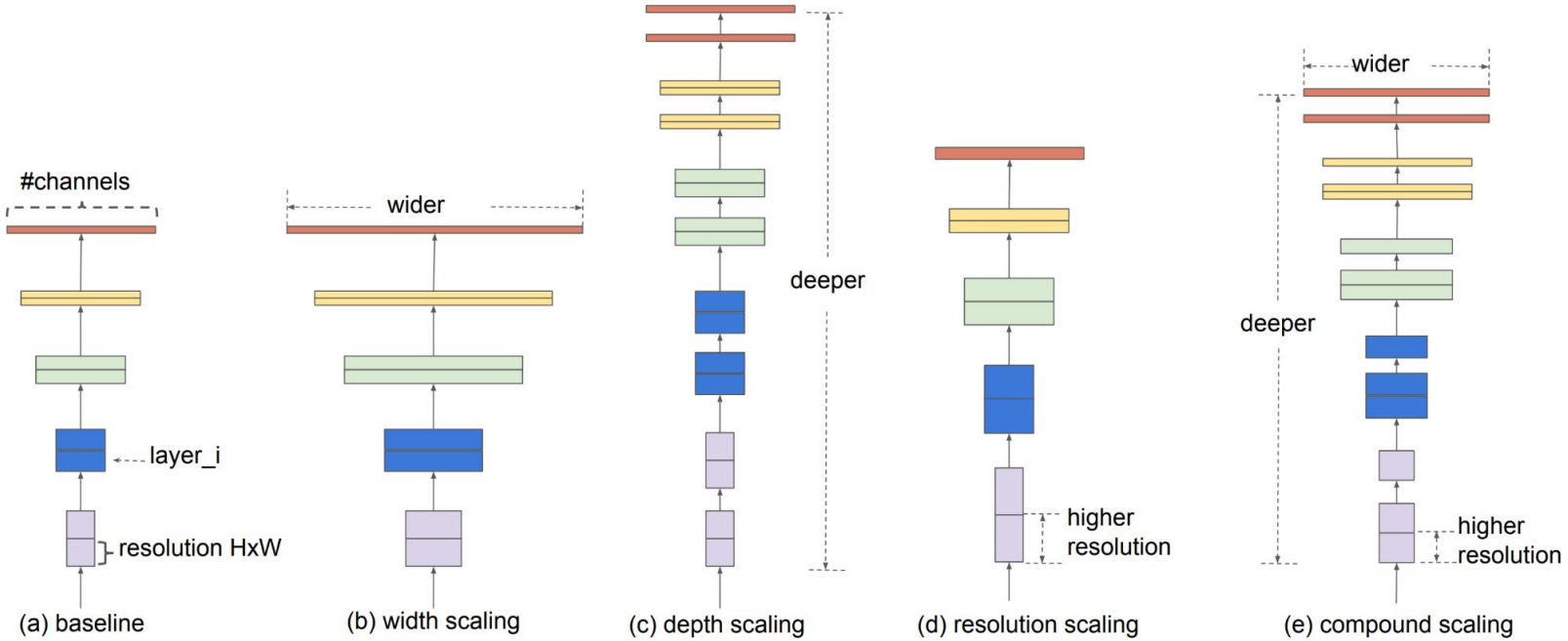


FUSION

EfficientNet

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (2019)

- Model Scaling: **width**, **depth**, **resolution**
- Compound scaling

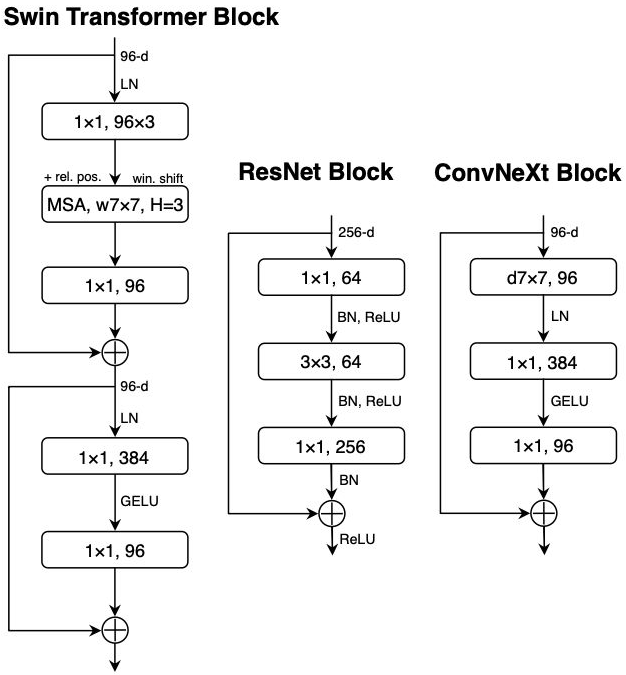


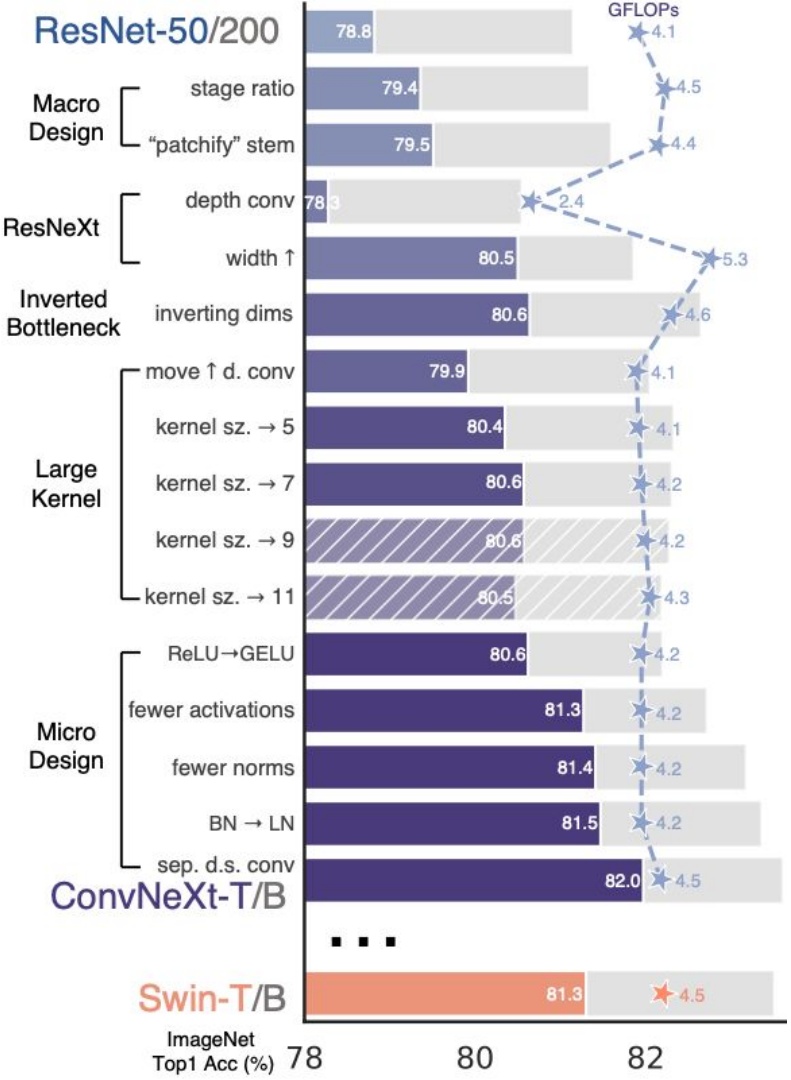
FUSION

ConvNeXt

[A ConvNet for the 2020s](#) (2022)

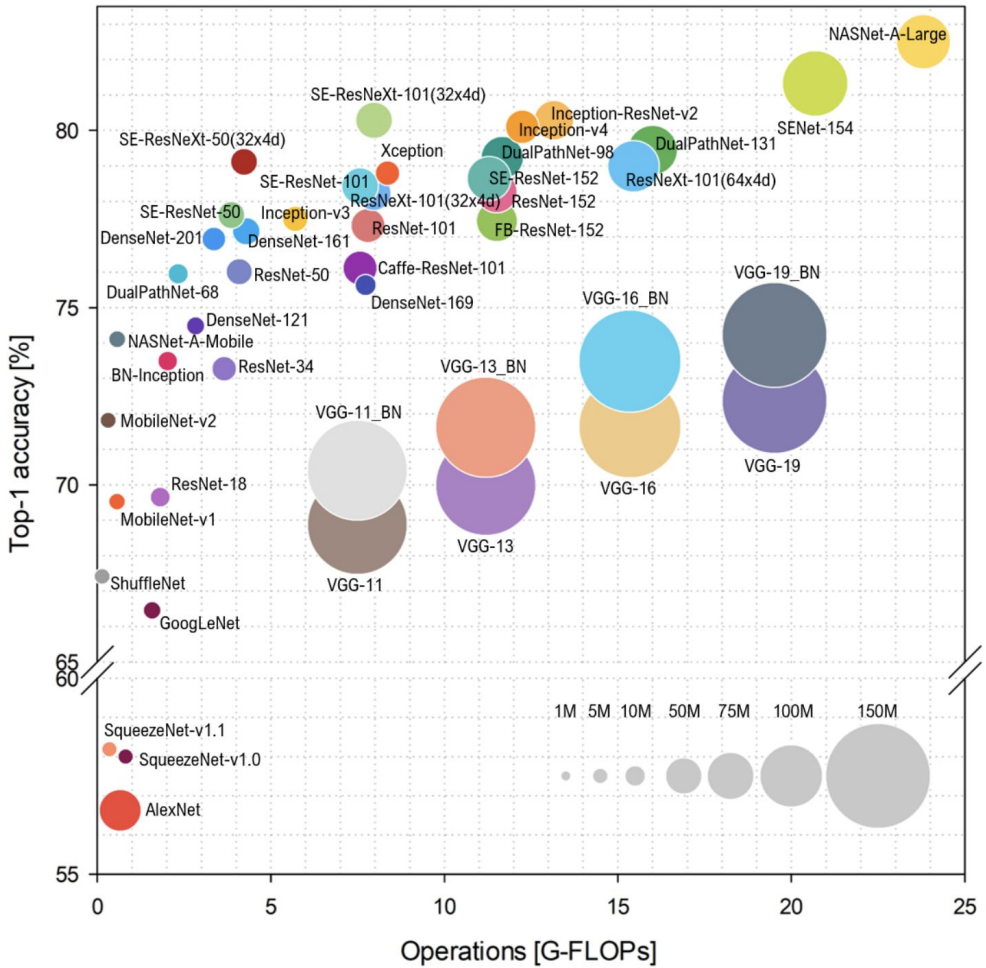
- 模仿Transformer模型設計





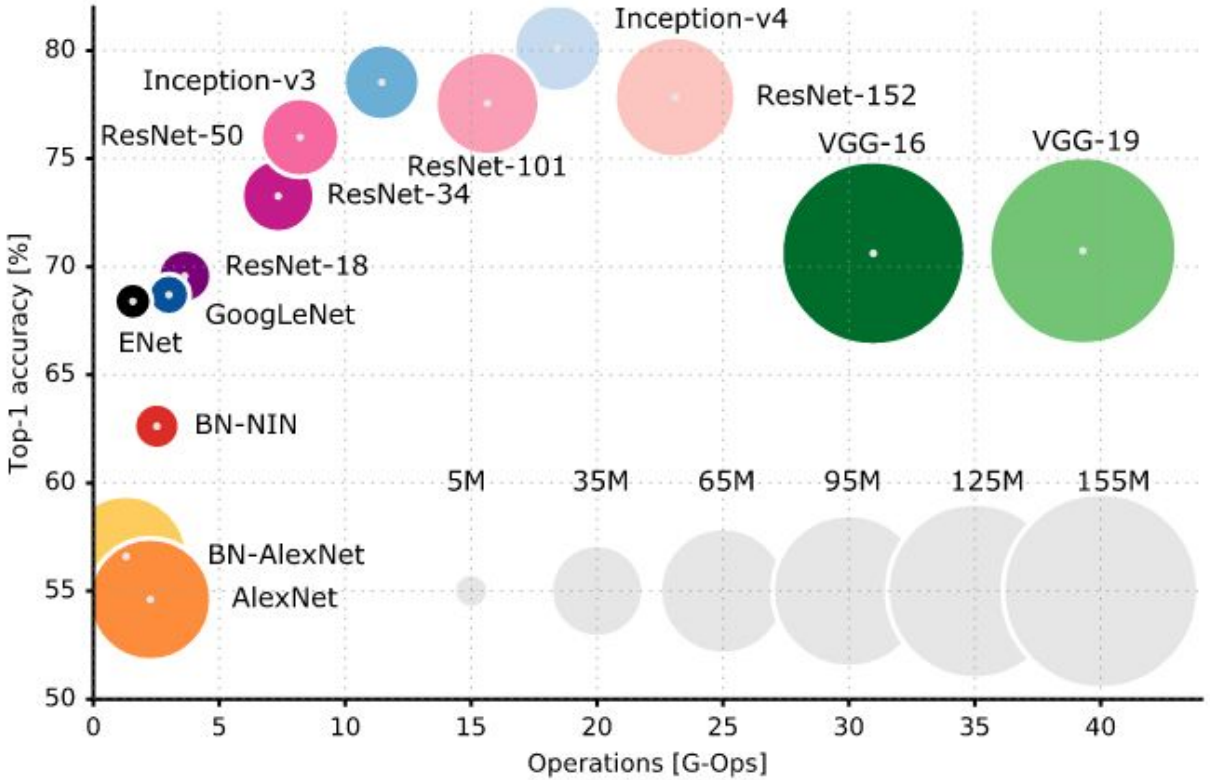
FUSION

CNN Bechmark



[Benchmark Analysis of Representative Deep Neural Network Architectures](#) (2018)

Performance v.s Computing Resource



source: https://p.migdal.pl/2017/04/30/teaching-deep-learning.html

PyTorch Vision Models

- Torchvision models list: https://pytorch.org/vision/stable/models.html

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNetV2
- MobileNetV3
- ResNeXt
- Wide ResNet
- MNASNet

```
import torchvision.models as models
resnet18 = models.resnet18()
alexnet = models.alexnet()
vgg16 = models.vgg16()
squeezenet = models.squeezenet1_0()
densenet = models.densenet161()
inception = models.inception_v3()
googlenet = models.googlenet()
shufflenet = models.shufflenet_v2_x1_0()
mobilenet_v2 = models.mobilenet_v2()
mobilenet_v3_large = models.mobilenet_v3_large()
mobilenet_v3_small = models.mobilenet_v3_small()
resnext50_32x4d = models.resnext50_32x4d()
wide_resnet50_2 = models.wide_resnet50_2()
mnasnet = models.mnasnet1_0()
```

FUSION

PyTorch Image Models (timm)

https://github.com/rwightman/pytorch-image-models

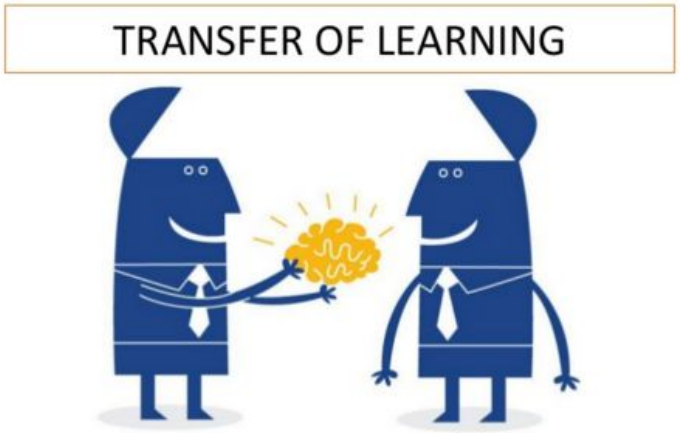
More Models

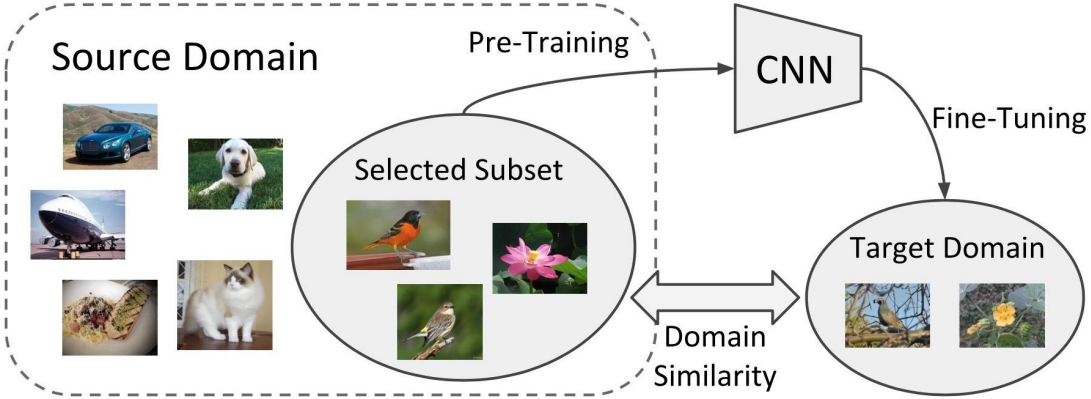
- Paper with Code: https://paperswithcode.com/
  - https://paperswithcode.com/sota/image-classification-on-imagenet
- model zoo: https://modelzoo.co/

FUSION

Transfer Learning(遷移學習)

- Not easy to have a dataset of sufficient size to train from scratch

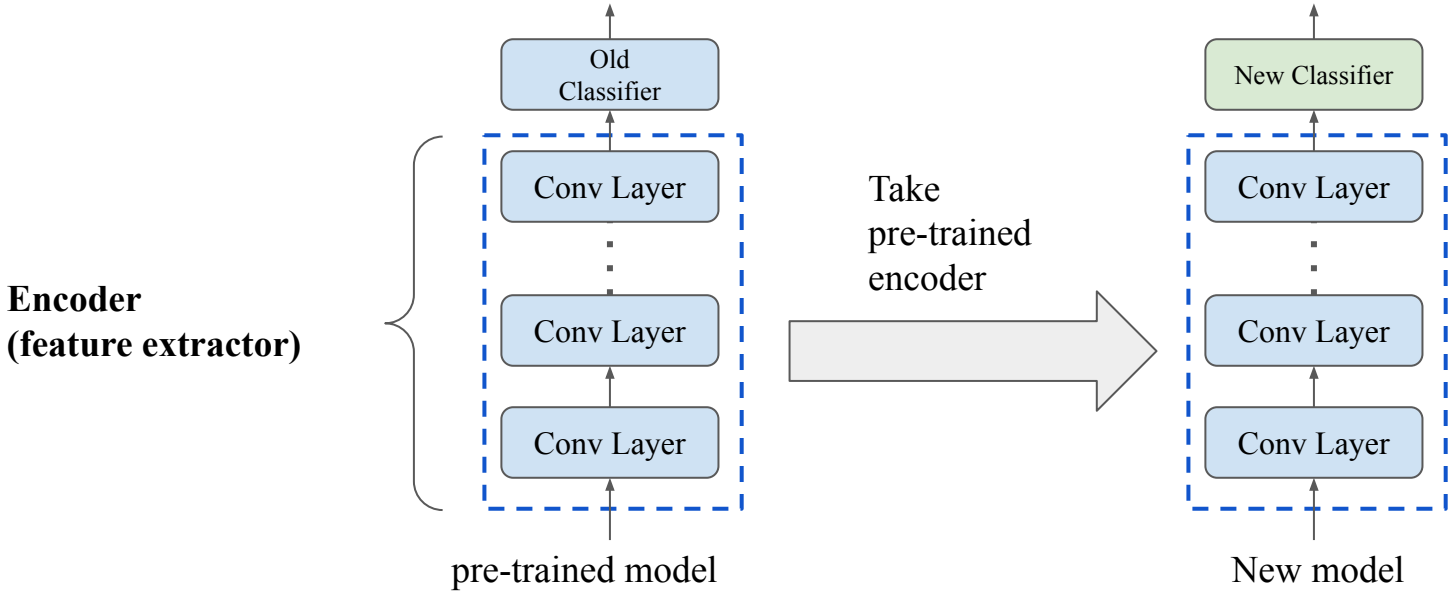




source
source

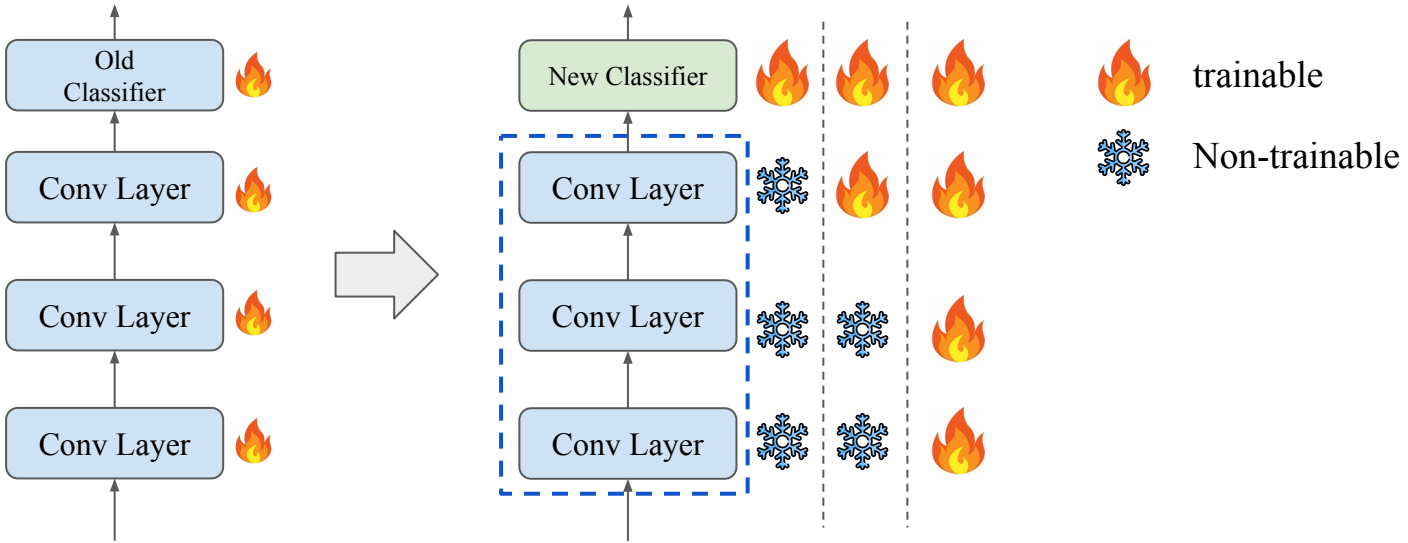
Transfer Learning

- Finetuning model
- ConvNet as fixed feature extractor
- Replace with “New” classifier



Transfer Learning

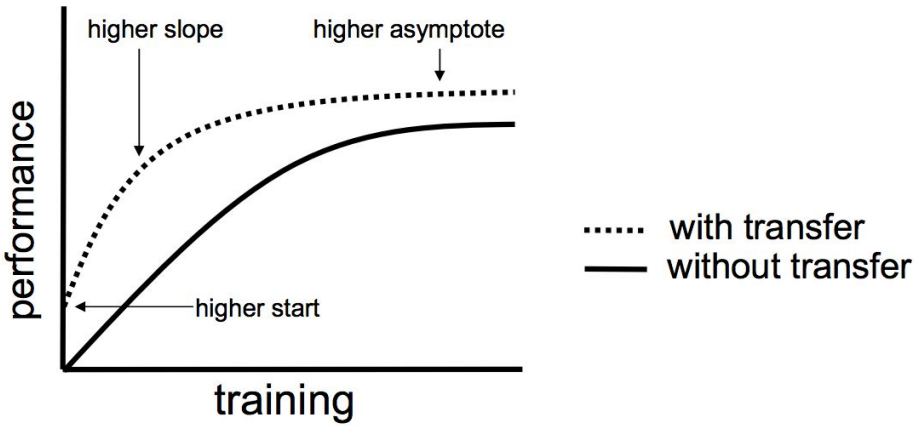
- **Freeze**: make model parameters **non-trainable**
- Less trainable parameters → Less data needed
- Better initialized parameters



Transfer Learning

1. **Build a model**
2. **Load pre-trained weights**
3. **Finetune**

- Take feature extractor
- Fine-tuning the Conv layers
  - **small** dataset, **similar**: simple classifier
  - **large** dataset, **similar**: fine tune all layers
  - **small** dataset, **different**: simple classifier from earlier layers
  - **large** dataset, **different**: take init weight only, tune all layers

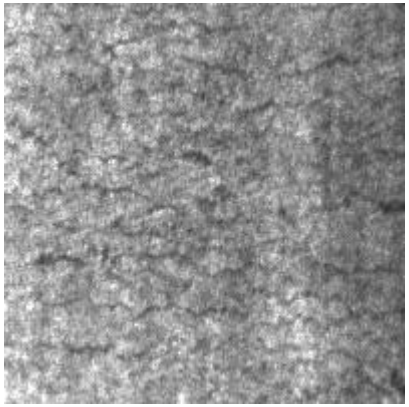


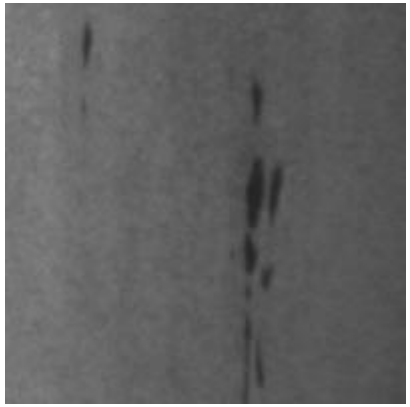
reference

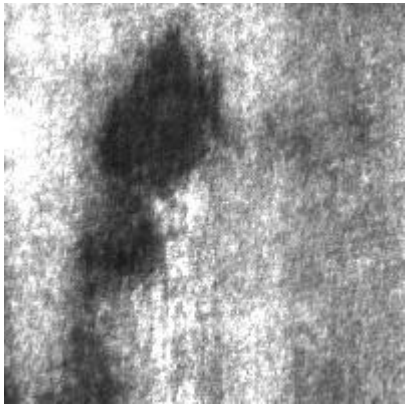
總結

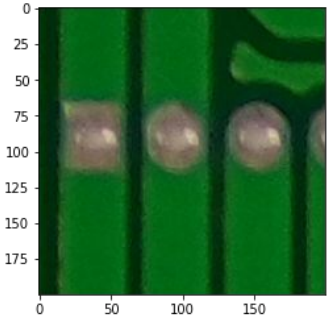
- 經典卷積神經網路
  - AlexNet
  - VGG
  - Inception
  - ResNet
  - Squeeze-Excitation Net
- 遷移學習
  - 介紹
  - 優點
  - 使用方法

Exercise: Defect Classification





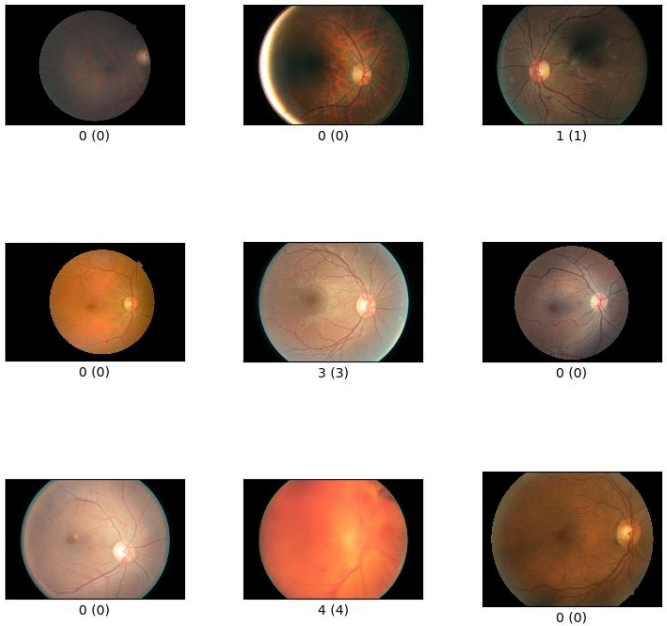




FUSION

Exercise & HW: Transfer Learning

- Kaggle:
  - https://www.kaggle.com/c/diabetic-retinopathy-classification-3



FUSION

FusionLab

https://github.com/taipingeric/fusionlab


```
pip install fusionlab
```

```
import fusionlab as fl

# Pytorch
encoder = fl.encoders.VGG16()
# Tensorflow
encoder = fl.encoders.TFVGG16()
```

fl.encoders (PyTorch, TF)

- AlexNet
- VGG16, VGG19
- InceptionNetV1
- ResNet50V1
- …